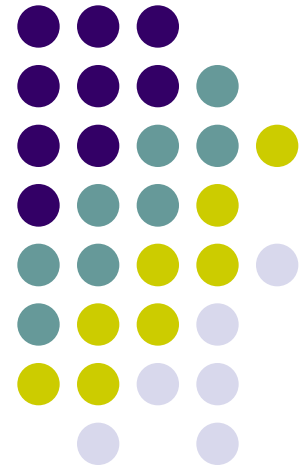
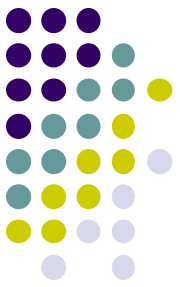


ממשק משתמש גרפי בעזרת SWT

תוכנה 1 בשפת Java
יעל אמסטרדמר ואלכסיי זגלסקי

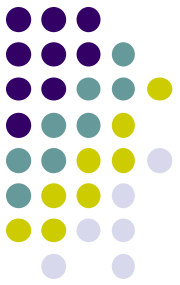




מה עושים היום?

- hashCode - equals
- ממשק משתמש גרפי

תזכורת: המחלקה Object



```
package java.lang;

public class Object {
    public final native Class<?> getClass();

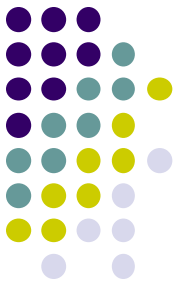
    public native int hashCode();

    public boolean equals(Object obj) {
        return (this == obj);
    }

    protected native Object clone() throws CloneNotSupportedException;

    public String toString() {
        return getClass().getName() + "@" +
            Integer.toHexString(hashCode());
    }
    ...
}
```

מה יודפס?

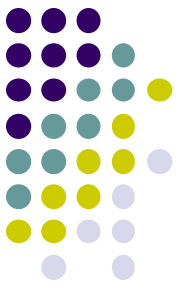


```
public class Name {  
    private String first, last;  
    ...
```

```
    public static void main(String[] args) {  
        Name name1 = new Name("Mickey", "Mouse");  
        Name name2 = new Name("Mickey", "Mouse");  
        System.out.println(name1.equals(name2));  
  
        List<Name> names = new ArrayList<Name>();  
        names.add(name1);  
        System.out.println(names.contains(name2));  
    }  
}
```

false

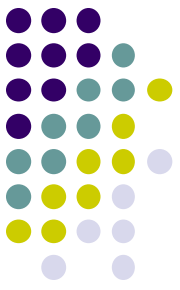
false



הבעיה

- רצינו השוואה לפי תוכן אבל לא דרסנו את equals
- מימוש ברירת המחדל הוא השוואה של מצביעים

```
public class Object {  
    ...  
    public boolean equals(Object obj) {  
        return (this == obj);  
    }  
    ...  
}
```



החזקה של equals

● רפלקסיבי

● `x.equals(x)` יחזיר `true`

● סימטרי

● `x.equals(y)` יחזיר `true` אם `y.equals(x)` יחזיר `true`

● טרנזיטיבי

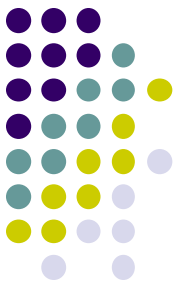
● אם `x.equals(y)` ו-`y.equals(z)` יחזיר `true` גם `x.equals(z)` יחזיר `true`
אז `x.equals(z)`

● עקבי

● סדרת קריאות ל-`x.equals(y)` תחזיר `true` (או `false`) באופן עקבי אם מידע שדרוש לצורך ההשוואה לא השתנה

● השוואה ל `null`

● `x.equals(null)` תמיד יחזיר `false`



מתכון ל equals

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null)  
        return false;  
    if (getClass() != obj.getClass())  
        return false;  
    Name other = (Name) obj;  
    return first.equals(other.first) &&  
        last.equals(other.last);  
}
```

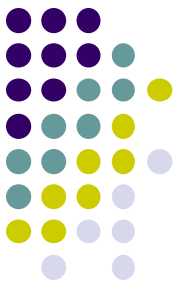
1. ודאו כי הארגומנט אינו מצביע לאובייקט הנוכחי

2. ודאו כי הארגומנט אינו null

3. ודאו כי הארגומנט הוא הטיפוס המתאים להשוואה

4. המירו את הארגומנט לטיפוס הנכון

5. לכל שדה "משמעותי", בידקו ששדה זה בארגומנט תואם לשדה באובייקט הנוכחי

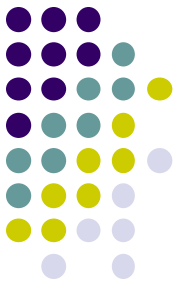


טעות נפוצה

- להגדיר את הפונקציה equals כך:

```
public boolean equals(Name name) {  
    return first.equals(other.first) &&  
        last.equals(other.last);  
}
```

- זו אינה דריסה (overriding) אלא העמסה (overloading)
- שימוש ב @Override יפתור את הבעיה



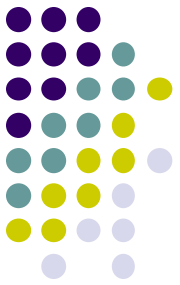
אז הכל בסדר?

```
public class Name {  
    ...  
    @Override public equals(Object obj) {  
        ...  
    }  
  
    public static void main(String[] args) {  
        Name name1 = new Name("Mickey", "Mouse");  
        Name name2 = new Name("Mickey", "Mouse");  
        System.out.println(name1.equals(name2));  
  
        List<Name> names = new ArrayList<Name>();  
        names.add(name1);  
        System.out.println(names.contains(name2));  
    }  
}
```

true יודפס

true יודפס

כמעט

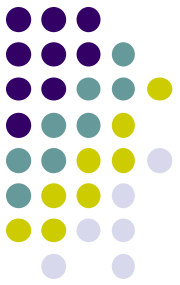


```
public class Name {  
    ...  
    @Override public equals(Object obj) {  
        ...  
    }  
  
    public static void main(String[] args) {  
        Name name1 = new Name("Mickey", "Mouse");  
        Name name2 = new Name("Mickey", "Mouse");  
        System.out.println(name1.equals(name2));  
  
        Set<Name> names = new HashSet<Name>();  
        names.add(name1);  
        System.out.println(names.contains(name2));  
    }  
}
```

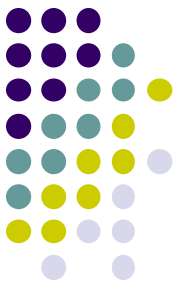
true טפדו

false טפדו

hashCode | equals



חובה לדרוס את hashCode בכל מחלקה
שדורסת את equals!



החזרה של hashCode

● עקביות

- מחזירה אותו ערך עבור כל הקריאות באותה ריצה, אלא אם השתנה מידע שבשימוש בהשוואת **equals** של המחלקה

● שוויון

- אם שני אובייקטים שווים לפי הגדרת **equals** אזי **hashCode** תחזיר ערך זהה עבורם

● חוסר שוויון

- אם שני אובייקטים אינם שווים לפי **equals** לא מובטח ש **hashCode** תחזיר ערכים שונים
- החזרת ערכים שונים יכולה לשפר ביצועים של מבני נתונים המבוססים על hashing (לדוגמא, **HashMap** ו **HashSet**)



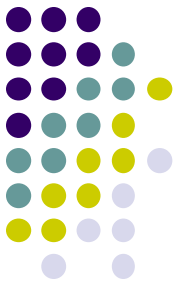
מימוש hashCode

```
@Override public int hashCode() {  
    return 31 * first.hashCode() + last.hashCode();  
}
```

- השתדלו לייצר hash כך שלאובייקטים שונים יהיה ערך hash שונה

- המימוש החוקי הגרוע ביותר (לעולם לא לממש כך!)

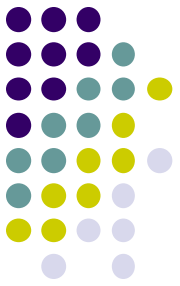
```
@Override public int hashCode() {  
    return 42;  
}
```



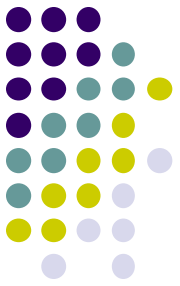
תמיכה באקליפס

- אקליפס תומך ביצירה אוטומטית (ומשולבת) של `hashCode` ו `equals`
- בתפריט Source ניתן למצוא `Generate hashCode() and equals()`

SWT



- בנויה על העיקרון של publish/subscribe
- אלמנטים בסיסיים (Widgets) מייצרים אירועים (Events) שאליהם נרשמים מאזינים (Listener)
- ה Widgets וה- Events מוגדרים ע"י כותבי הספרייה
- מאזינים נכתבים ע"י המשתמש
- תגובות שונות לאירועים זהים כתלוי באפליקציה



SWT Widgets

- אבני הבניין של ממשקים גרפיים
- מוגדרים ב org.eclipse.swt.widgets
- תת-טיפוסים של המחלקה האבסטרקטית [Widget](#)



Shell

Jack and Jill went up
the hill to fetch a pail
of water, Jack fell
down and broke his
crown and Jill came
tumbling after!

Label

The quick brown fox jum

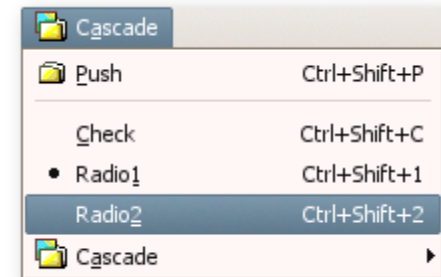
Text

Name	Type	Size
<input type="checkbox"/> Index:0	classes	0
<input checked="" type="checkbox"/> Index:1	databases	2556
<input type="checkbox"/> Index:2	images	9157
<input checked="" type="checkbox"/> Index:3	classes	0
<input type="checkbox"/> Index:4	databases	2556

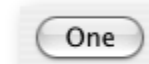
Table

- Apples
- Oranges
- Bananas
- Grapefruit
- Peaches
- Kiwi
- Apricots
- Strawberries
- The Longest String

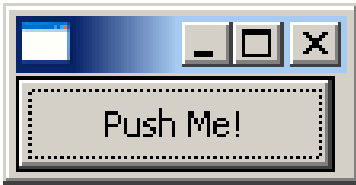
List



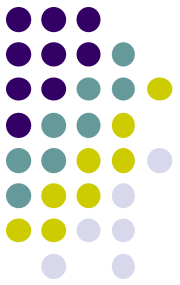
Menu



Button



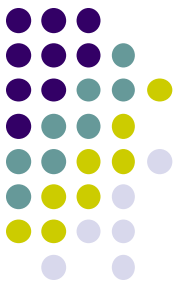
כפתור



```
public class ShellWithButton {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        Shell shell = new Shell (display);

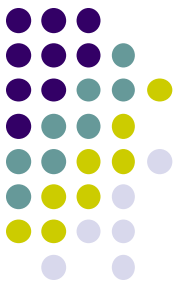
        Button ok = new Button (shell, SWT.PUSH);
        ok.setText ("Push Me!");
        ok.setLocation(0,0);
        ok.setSize(100,30);

        shell.pack ();
        shell.open ();
        while (!shell.isDisposed ()) {
            if (!display.readAndDispatch())
                display.sleep ();
        }
        display.dispose ();
    }
}
```



הוספת טיפול בארועים

- הכפתור לא מגיב ללחיצות. יש להוסיף טיפול בארוע "לחיצה"
- על המחלקה המטפלת לממש את המנשק `SelectionListener`
- על הכפתור עצמו להגדיר מי העצם (או העצמים) שיטפלו בארוע
- כמה גישות אפשריות:
 - הגדרת מחלקה שיורשת מכפתור
 - מחלקה שמכילה כפתור כאחד משדותיה
 - יצירת מחלקה עצמאית שתטפל באירועי הלחיצה
- לכל אחת מהאפשרויות יתרונות וחסרונות שידונו בהמשך



הוספת טיפול בארועים

- הכפתור לא מגיב ללחיצות. יש להוסיף טיפול באירוע "לחיצה"
- עלינו לממש מאזין המקבל שמטפל באירוע ולהרשם על הווידג'ט המתאים.
- כיצד נדע אילו אירועים מייצר ווידג'ט? איזה מנשק עלינו לממש?
- נסתכל בתיעוד

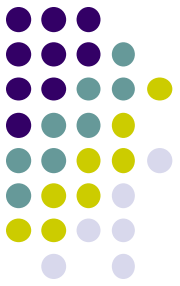
Events:
Selection

Method Summary

void [addSelectionListener](#) ([SelectionListener](#) listener)

Adds the listener to the collection of listeners who will be notified when the of the messages defined in the `SelectionListener` interface.

טיפול בארועים במחלקה נפרדת

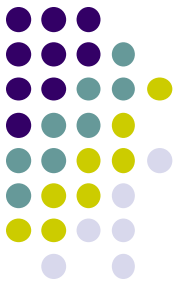


```
public class ButtonHandler
    implements SelectionListener {

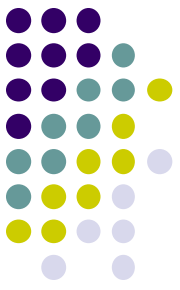
    public void widgetSelected(SelectionEvent e) {
        if (e.getSource() instanceof Button) {
            Button b = (Button) e.getSource();
            b.setText("Thanks!");
        }
    }

    public void widgetDefaultSelected(SelectionEvent e) {
        // TODO Auto-generated method stub
    }
}
```

טיפול בארועים במחלקה נפרדת

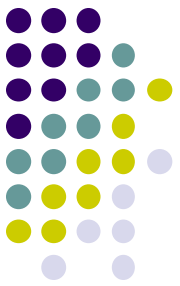


```
public class ShellWithButton {
    public static void main(String[] args) {
        Display display = Display.getDefault();
        Shell shell = new Shell (display);
        Button ok = new Button(shell, SWT.PUSH);
        ok.addSelectionListener(new ButtonHandler());
        ok.setText ("Push Me!");
        ok.setLocation(0,0);
        ok.setSize(100,30);
        shell.pack ();
        shell.open ();
        while (!shell.isDisposed ()) {
            if (!display.readAndDispatch ()) display.sleep ();
        }
        display.dispose ();
    }
}
```

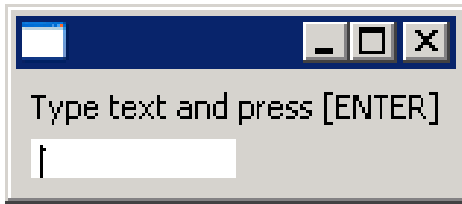


טיפול בארועים במחלקה נפרדת

- לעיתים הטיפול באירוע דורש הכרות אינטימית עם המקור (כדי להימנע מחשיפת המבנה הפנימי של המקור)
- שימוש במחלקה פנימית יוצר את האינטימיות הדרושה
- בדוגמא הבאה נרצה לעדכן תווית על סמך קלט מהמשתמש
- דרושה הכרות לא רק עם יוצר האירוע (Text) אלא גם עם חלקים אחרים במבנה



מחלקה פנימית

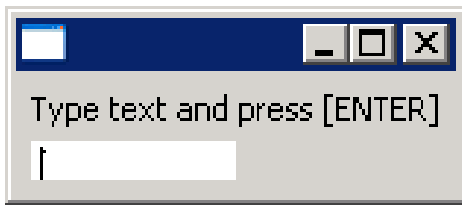


```
public class ShellWithLabelAndTextField {  
    private Label l;  
    private Text t;  
  
    public static void main(String[] args) { ...}  
    public void createShell() {...}
```

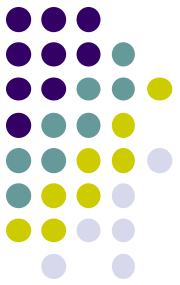
```
public class InnerHandler implements KeyListener  
{  
    public void keyPressed(KeyEvent e) {  
        if(e.character == SWT.CR){  
            l.setText(t.getText());  
            t.setText("");  
        }  
    }  
  
    public void keyReleased(KeyEvent e) {  
        // TODO Auto-generated method stub  
    }  
}
```

המחלקה הפנימית ניגשת לשדות הפרטיים של המחלקה העוטפת

```
}
```



מחלקה פנימית



```
public class ShellWithLabelAndTextField {

    private Label l;
    private Text t;

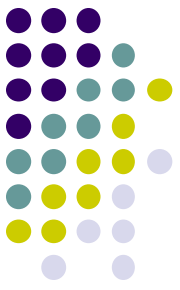
    public static void main(String[] args) {
        ShellWithLabelAndTextField shell = new ShellWithLabelAndTextField();
        shell.createShell();
    }
    public void createShell() {
        Display display = new Display ();
        Shell shell = new Shell (display);

        GridLayout gl = new GridLayout();
        shell.setLayout(gl);

        l = new Label (shell, SWT.CENTER);
        l.setText ("Type text and press [ENTER]");

        t = new Text(shell, SWT.LEFT);
        t.addListener(new InnerHandler());

        // pack(), open(), while ... Dispose()
    }
}
```

שימוש במחלקות אנונימיות

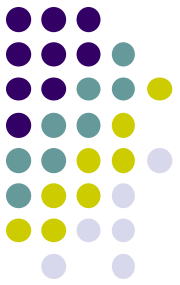
- בדרך כלל נזדקק רק למאזין יחיד לכל אירוע
- נשתמש במחלקה לוקאלית אנונימית

● תזכורת: `new className([argument-list]) {classBody}`

- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שיורשת באופן אוטומטי מ `className`

`new interfaceName() {classBody}`

- יצירת מופע חדש של מחלקה ללא שם, שטרם הוגדרה, שממשת באופן אוטומטי את `interfaceName`



מחלקה אנונימית

```
public class ShellWithLabelAndTextField {
```

```
...
```

```
public void createShell() {
```

```
...
```

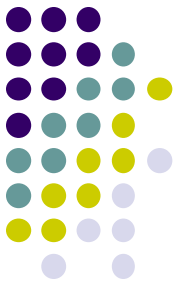
```
t.addListener(new KeyListener() {  
    public void keyPressed(KeyEvent e) {  
        if (e.character == NEW_LINE_CHAR) {  
            l.setText(t.getText());  
            t.setText("");  
        }  
    }  
  
    public void keyReleased(KeyEvent e) {  
        // TODO Auto-generated method stub  
    }  
});
```

← סוגר סוגריים של המתודה addKeyListener()

```
// pack(), open(), while ... Dispose()
```

```
}
```

```
}
```



שימוש ב Adapter

```
public class ShellWithLabelAndTextField {
```

```
...
```

```
public void createShell() {
```

```
...
```

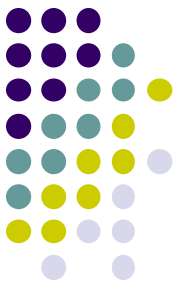
```
t.addKeyListener(new KeyAdapter() {  
    public void keyPressed(KeyEvent e) {  
        if (e.character == NEW_LINE_CHAR) {  
            l.setText(t.getText());  
            t.setText("");  
        }  
    }  
});
```

← סוגר סוגריים של המתודה addKeyListener()

```
// pack(), open(), while ... Dispose()
```

```
}
```

```
}
```



המחלקה SWT

- מוגדרת ב `org.eclipse.swt.SWT`
- אוסף של קבועים:
 - אירועים – `Activate`, `Close`, `FocusIn`, `MouseDown`, ...
 - צבעים – `COLOR_BLACK`, `COLOR_BLUE`, ...
 - תווים – `CR`, `DEL`, `ESC`, ...
 - אירוע מקשים – `ARROW_DOWN`, `END`, ...